Introduction to the Network Weather Service

T.L. Thomas, University of New Mexico

Feb 3, 2003

The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Rich Wolski, Neil Spring, and Jim Hayes, Journal of Future Generation Computing Systems,Volume 15, Numbers 5-6, pp. 757-768, October, 1999.

Monitoring, Prediction and Scheduling on the Grid

Jennifer M. Schopf

Argonne National Lab

Scheduling and Prediction on the Grid

- First step of Grid computing – basic functionality
  - Run my job
  - Transfer my data
  - Security
- Next step – more efficient use of the resources
  - Scheduling
  - Prediction
  - Monitoring

How can these resources be used effectively?

- Efficient scheduling
  - Selection of resources
  - Mapping of tasks to resources
  - Allocating data
- Accurate prediction of performance
- Good performance prediction modeling techniques

Replica Selection

- Why not use something like Network Weather Service (NWS) probes?
  - Wolski and Swany, UCSB
  - Logging and prediction
  - Small data transfers
  - CPU load, memory, etc.

The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Rich Wolski, Neil Spring, and Jim Hayes, Journal of Future Generation Computing Systems,Volume 15, Numbers 5-6, pp. 757-768, October, 1999.

Paper Outline:

The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing

Rich Wolski [a,1] Neil T. Spring [b,2] Jim Hayes [b,3]

[a] *University of California, San Diego and University of Tennessee, Knoxville*

[b] *University of California, San Diego*

Abstract

The goal of the Network Weather Service is to provide accurate forecasts of dynamically changing performance characteristics from a distributed set of metacomputing resources. Providing a ubiquitous service that can both track dynamic performance changes and remain stable in spite of them requires adaptive programming techniques, an architectural design that supports extensibility, and internal abstractions that can be implemented efficiently and portably. In this paper, we describe the current implementation of the NWS for Unix and TCP/IP sockets and provide examples of its performance monitoring and forecasting capabilities.

In this paper we describe the latest implementation of the **Network Weather Service** (NWS), a distributed, generalized system for producing short-term performance forecasts based on historical performance measurement. The goal of the system is to dynamically characterize and forecast the performance deliverable at the application level from a set of network and computational resources. Such forecasts have been used successfully to implement dynamic scheduling agents for metacomputing applications [26,3], and to choose between replicated web pages [1].

The implementation of the NWS relies on adaptivity to enable stability, accuracy, non-intrusiveness, and extensibility.

The NWS is designed to maximize four possibly conflicting functional characteristics. It must meet these goals despite the highly dynamic execution environment and evolving software infrastructure provided by shared metacomputing systems [2].

- **Predictive Accuracy**: The NWS must be able to provide accurate estimations of future resource performance in a timely manner.
- **Non-intrusiveness**: The system must load the resources it is monitoring as little as possible.
- **Execution longevity**: To be effective, the NWS should be available at any time as a general system service. It should not execute and complete – its execution lifetime is logically indefinite.
- **Ubiquity**: As a system service, the NWS should be available from all potential execution sites within a resource set. Similarly, it should be able to monitor and forecast the performance of all available resources.

We have constructed the current NWS using using four different *component processes*.

- **Persistent State** process: stores and retrieves measurements from persistent storage.
- **Name Server** process: implements a directory capability used to bind process and data names with low-level contact information (e.g. TCP/IP port number, address pairs).
- **Sensor** process: gathers performance measurements from a specified resource.
- **Forecaster** process: produces a predicted value of deliverable performance during a specified time frame for a specified resource.

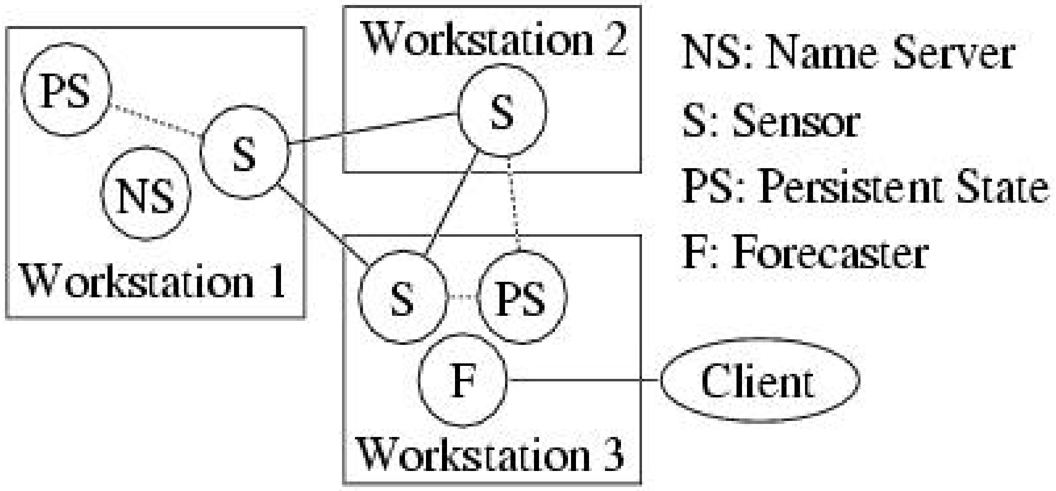


Fig. 1. NWS Processes distributed across three workstations. The Name Server resides on only one host in the system. Sensors monitor the performance characteristics of networks and processors and send their measurements to Persistent State managers. The Forecaster acts as a proxy for application scheduling clients and user queries. Workstation 2 can be integrated in the system without any associated storage space, since its persistent state is managed on Workstation 3.

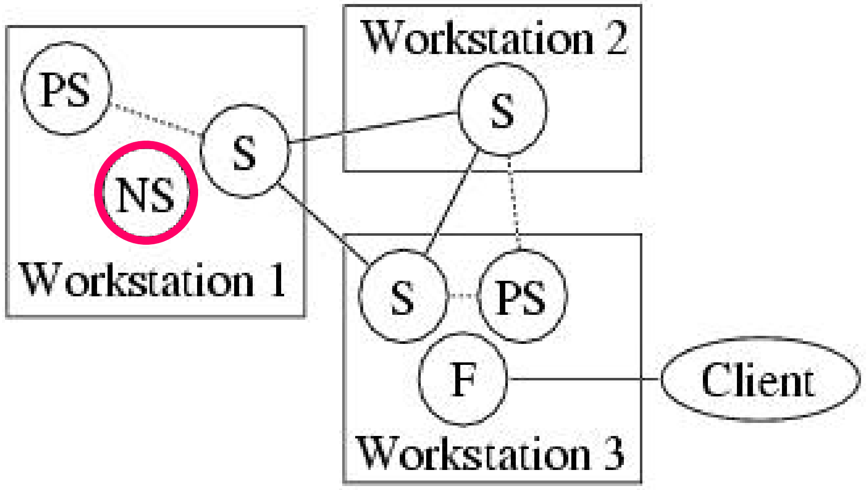


Fig. 1. NWS Processes distributed across three workstations. The Name Server resides on only one host in the system. Sensors monitor the performance characteristics of networks and processors and send their measurements to Persistent State managers. The Forecaster acts as a proxy for application scheduling clients and user queries. Workstation 2 can be integrated in the system without any associated storage space, since its persistent state is managed on Workstation 3.

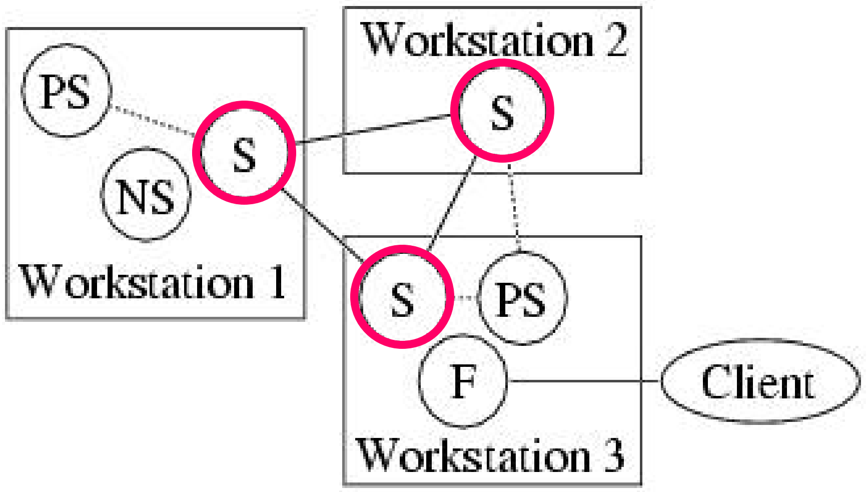


Fig. 1. NWS Processes distributed across three workstations. The Name Server resides on only one host in the system. Sensors monitor the performance characteristics of networks and processors and send their measurements to Persistent State managers. The Forecaster acts as a proxy for application scheduling clients and user queries. Workstation 2 can be integrated in the system without any associated storage space, since its persistent state is managed on Workstation 3.

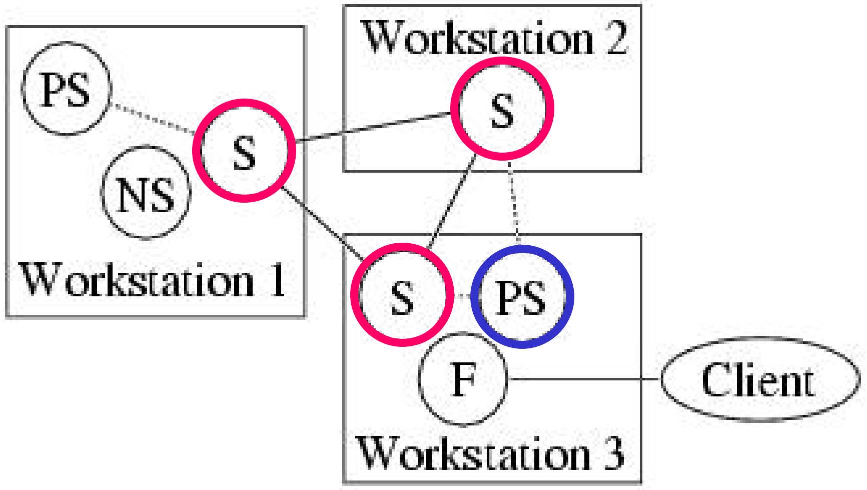
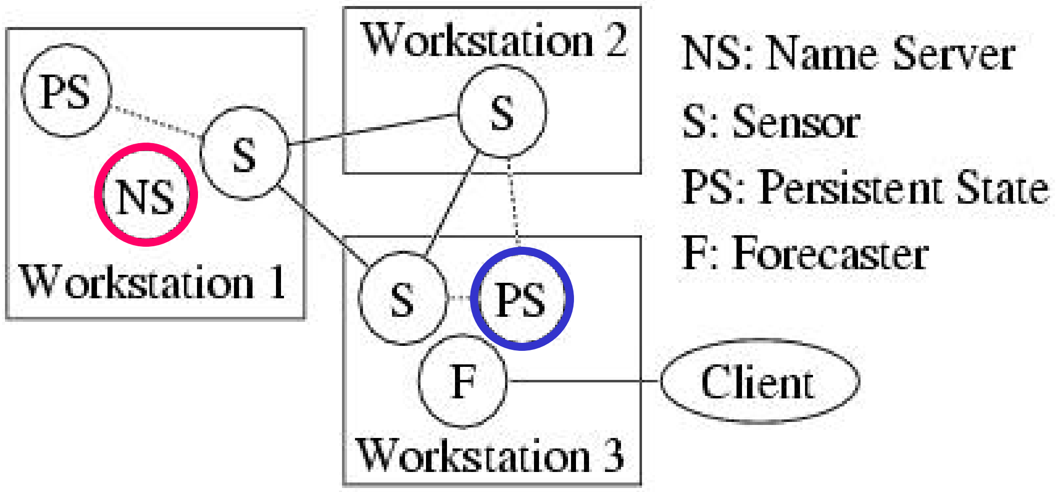


Fig. 1. NWS Processes distributed across three workstations. The Name Server resides on only one host in the system. Sensors monitor the performance characteristics of networks and processors and send their measurements to Persistent State managers. The Forecaster acts as a proxy for application scheduling clients and user queries. Workstation 2 can be integrated in the system without any associated storage space, since its persistent state is managed on Workstation 3.



The location of the Persistent State process that a Sensor will use for each of the measurements it gathers is specified when the Sensor is configured. When it is initialized, each Sensor registers the location of the Persistent State process that stores its measurement data with the Name Service so that measurement data may be located by name.

At present, the Name Server process ~~that implements this functionality~~ is based on the more general Persistent State process.

We are converting the Name Service to use an implementation of the Lightweight Directory Access Protocol [32] (LDAP).

We anticipate that state storage and name service functionality will eventually be provided by lower-level metacomputing services, such as the Globus Metacomputing Directory Service [10] and the Legion Resource Directory Service [8].

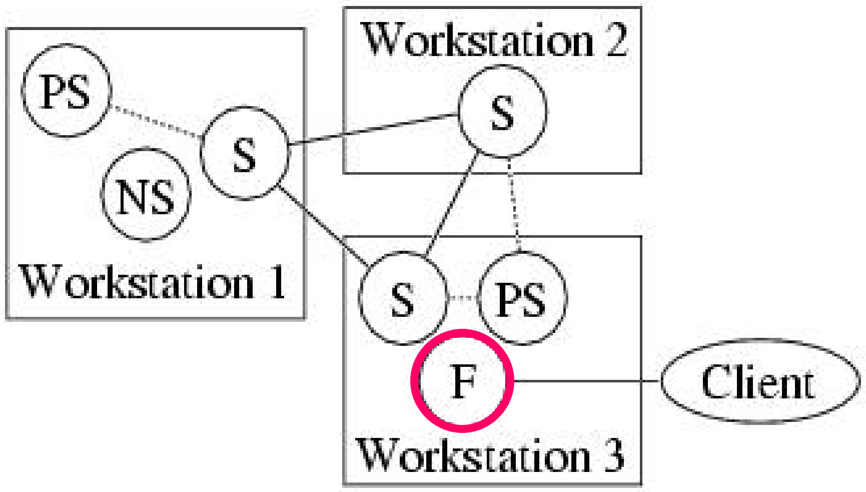


Fig. 1. NWS Processes distributed across three workstations. The Name Server resides on only one host in the system. Sensors monitor the performance characteristics of networks and processors and send their measurements to Persistent State managers. The Forecaster acts as a proxy for application scheduling clients and user queries. Workstation 2 can be integrated in the system without any associated storage space, since its persistent state is managed on Workstation 3.

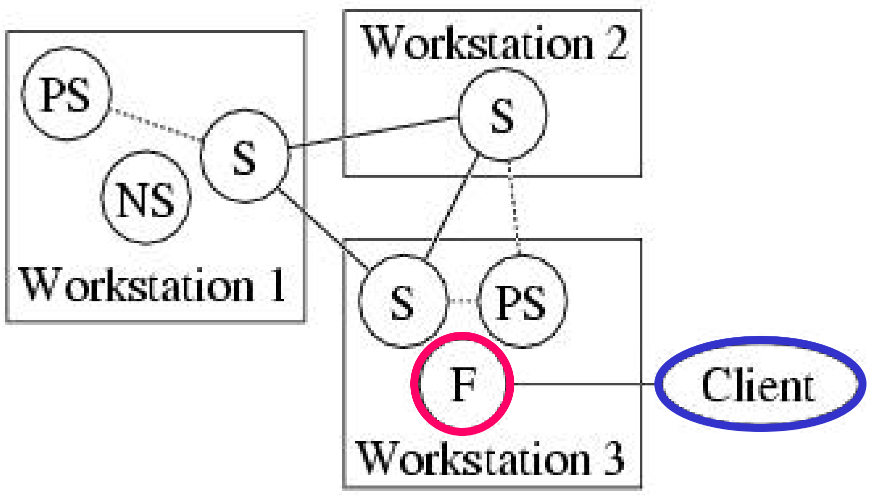


Fig. 1. NWS Processes distributed across three workstations. The Name Server resides on only one host in the system. Sensors monitor the performance characteristics of networks and processors and send their measurements to Persistent State managers. The Forecaster acts as a proxy for application scheduling clients and user queries. Workstation 2 can be integrated in the system without any associated storage space, since its persistent state is managed on Workstation 3.

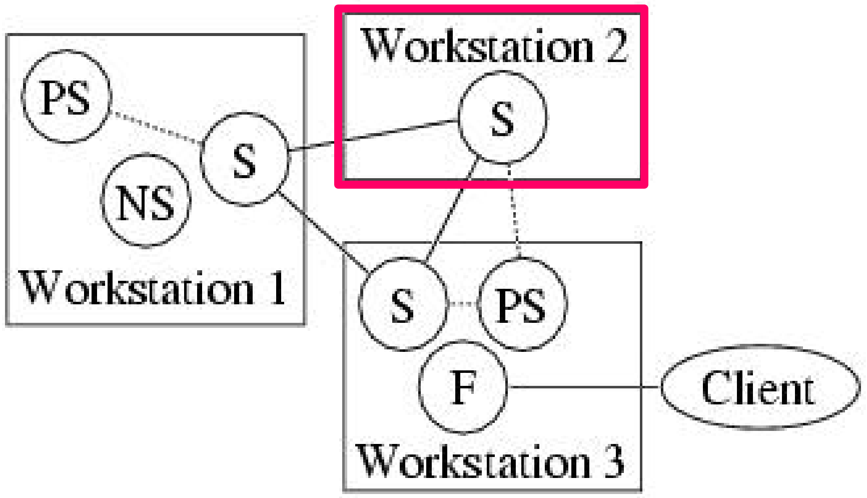


Fig. 1. NWS Processes distributed across three workstations. The Name Server resides on only one host in the system. Sensors monitor the performance characteristics of networks and processors and send their measurements to Persistent State managers. The Forecaster acts as a proxy for application scheduling clients and user queries. Workstation 2 can be integrated in the system without any associated storage space, since its persistent state is managed on Workstation 3.

4. Performance Monitoring: NWS Sensors

In general, there is a tension between the intrusiveness of a monitoring technique and the measurement accuracy it provides. The NWS attempts to use both extant performance monitoring utilities and active resource occupancy to measure performance.

The function of an NWS *Sensor* is to gather and store time stamp-performance measurement pairs for a specific resource. Each Sensor process may measure several different performance characteristics of the resource it is sensing. The TCP/IP network Sensor, for example, provides both bandwidth and end-to-end round-trip latency measurements, but each set of measurements is named and stored separately. That is, a Sensor does not attempt to correlate the separate performance characteristics of a resource it monitors.

To make the system more robust, all NWS processes are stateless. Persistent state – state that must be able to survive the failure of a process' memory – is managed explicitly throughout the system using Persistent State processes. Each Persistent State process provides a simple text string storage and retrieval service and allows each stored string to be associated with an optional time stamp. Each storage or retrieval request must be accompanied by the name of the data set that is to be accessed, and any data that is sent to a Persistent State process is immediately written to disk before an acknowledgement is returned. Since the function of the NWS is to generate forecasts which lose their utility after their epoch passes, the system does not maintain any data indefinitely. Each file that a Persistent State process uses is managed as a circular queue, the length of which is a configuration option. Data to be archived indefinitely must be fetched and stored in some more permanent medium outside the NWS before the queue fills.

CPU Sensor

The NWS CPU Sensor combines information from Unix system utilities *uptime* and *vmstat* with periodic active CPU occupancy tests to provide measurements of CPU availability on timeshared Unix systems.

The CPU Sensor uses the one-minute measurement to calculate the fraction of the CPU occupancy time that a process would get if it were to run at the moment the *uptime* measurement were taken. From *vmstat* output, the CPU Sensor uses a combination of idle time, user time, and system time measurements to generate an estimate of the available CPU occupancy fraction [31].

neither *uptime* nor *vmstat* provides information on the priority of processes presently running on the system. In order to obtain more accurate measurements, the CPU Sensor incorporates active probes into its calculations. It periodically runs an artificial, compute-intensive "probe" program and calculates the CPU availability as the ratio of its observed CPU occupancy time to the wall-clock time of its execution.

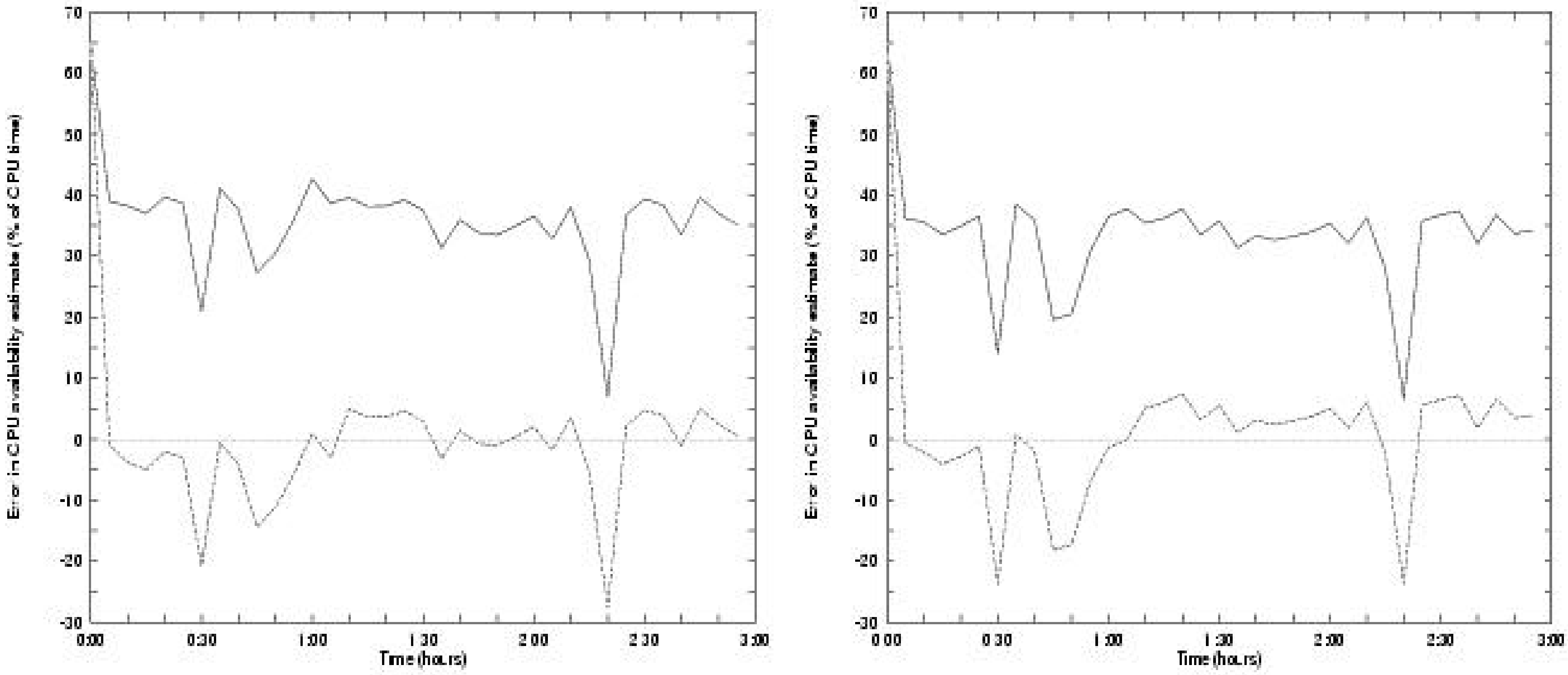
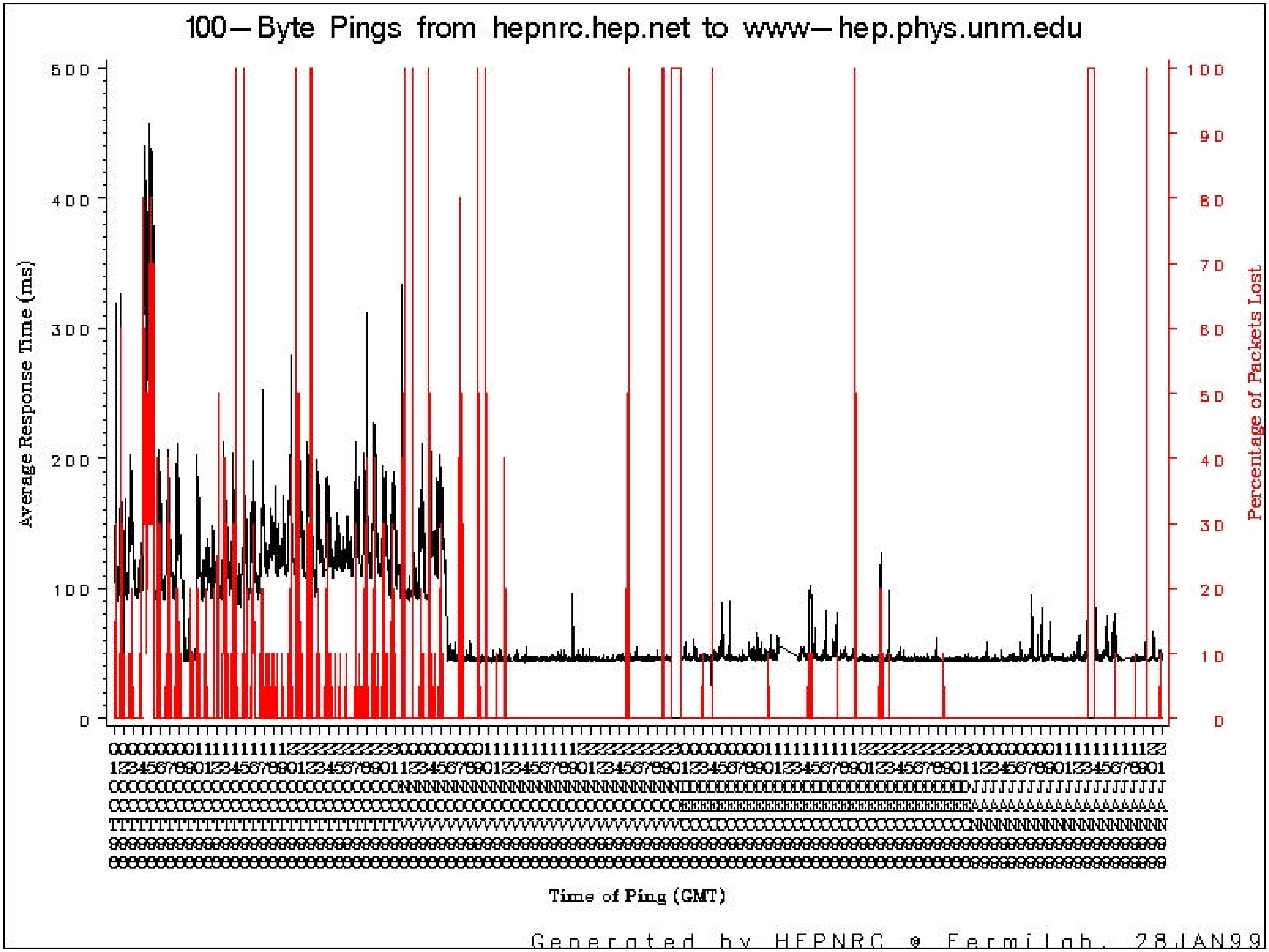


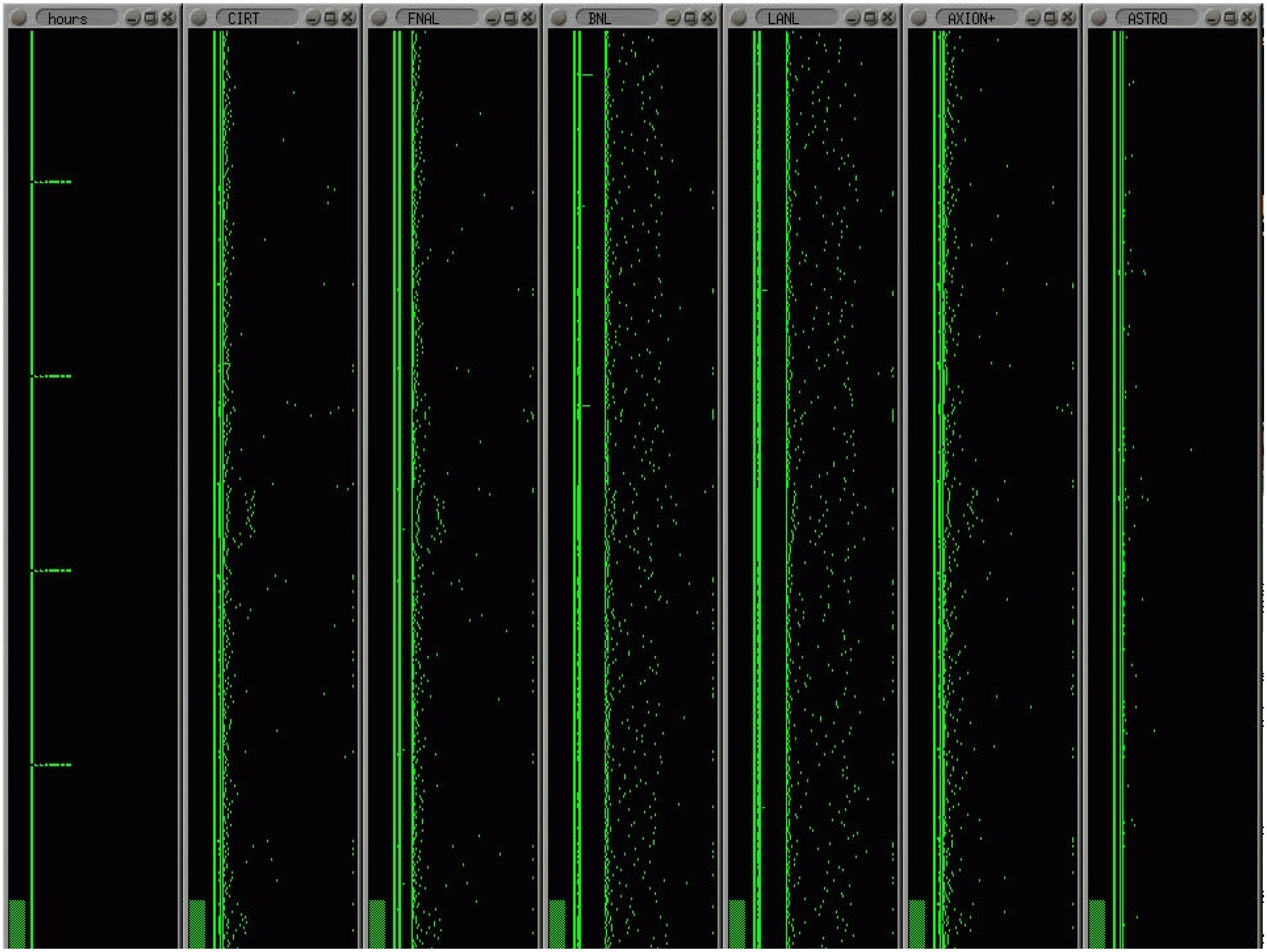
Fig. 2. Improvement from active probing in estimates of CPU availability generated using *uptime* (left) and *vmstat* (right). The solid line shows the amount of error in unadjusted estimates; the dashed line the error in adjusted estimates.

Network Sensor

Because end-to-end network performance data between arbitrary machines is not consistently available, NWS network Sensors rely on active network probes exclusively when determining network load. Each probe consists of a timed network operation.

Currently, the NWS network Sensor is capable of measuring three network performance characteristics: small-message round-trip time, large-message throughput, and TCP socket connect-disconnect time. The small-message probe consists of a 4-byte TCP socket transfer that is timed as it is sent from a source Sensor to a destination Sensor and back. The socket connection used to facilitate the transfer is already established before the probe is conducted. Large-message throughput (that is taken to measure available network bandwidth at the application level) is calculated by timing the transfer of a message using TCP and the acknowledgement of its receipt by the receiving sensor.

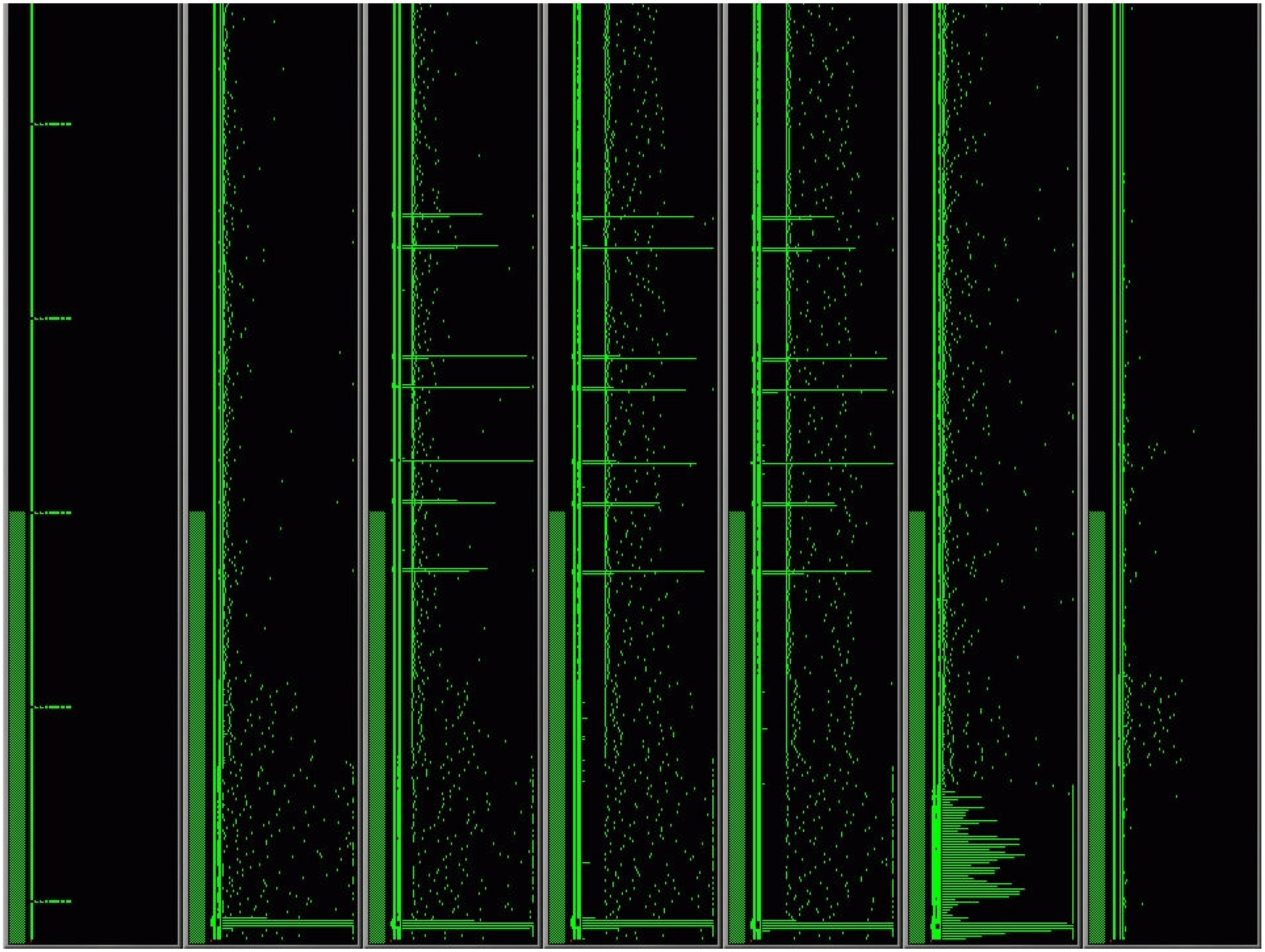

100–Byte Pings from hepnrc.hep.net to www–hep.phys.unm.edu
Average Response Time (ms)
Percentage of Packets Lost
0
100
200
300
400
500
10
20
30
40
50
60
70
80
90
Time of Ping (GMT)
Generated by HEPNRC @ Fermilab, 28JAN99

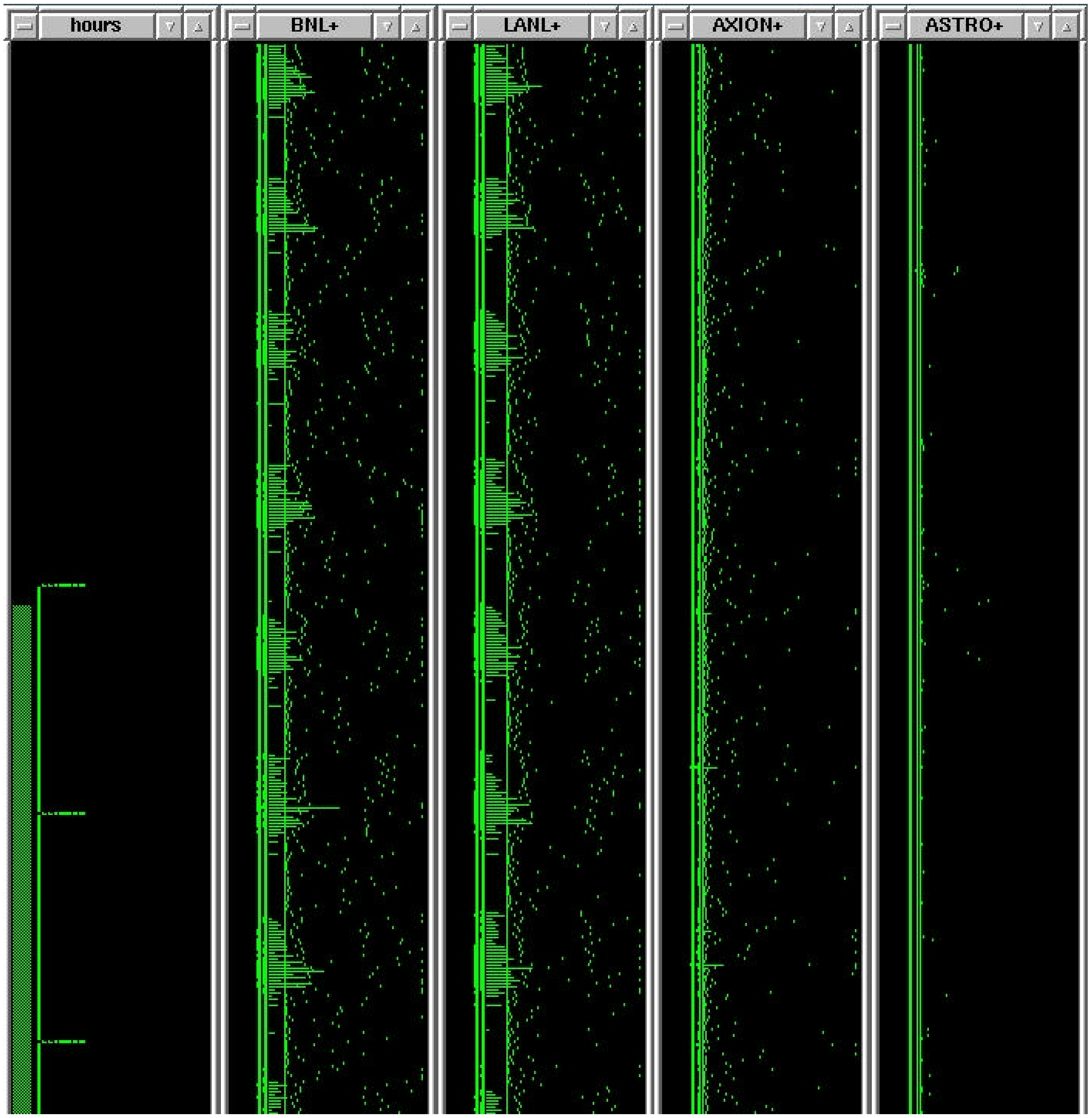

hours
CIRT
FNAL
BNL
LANL
AXION+
ASTRO

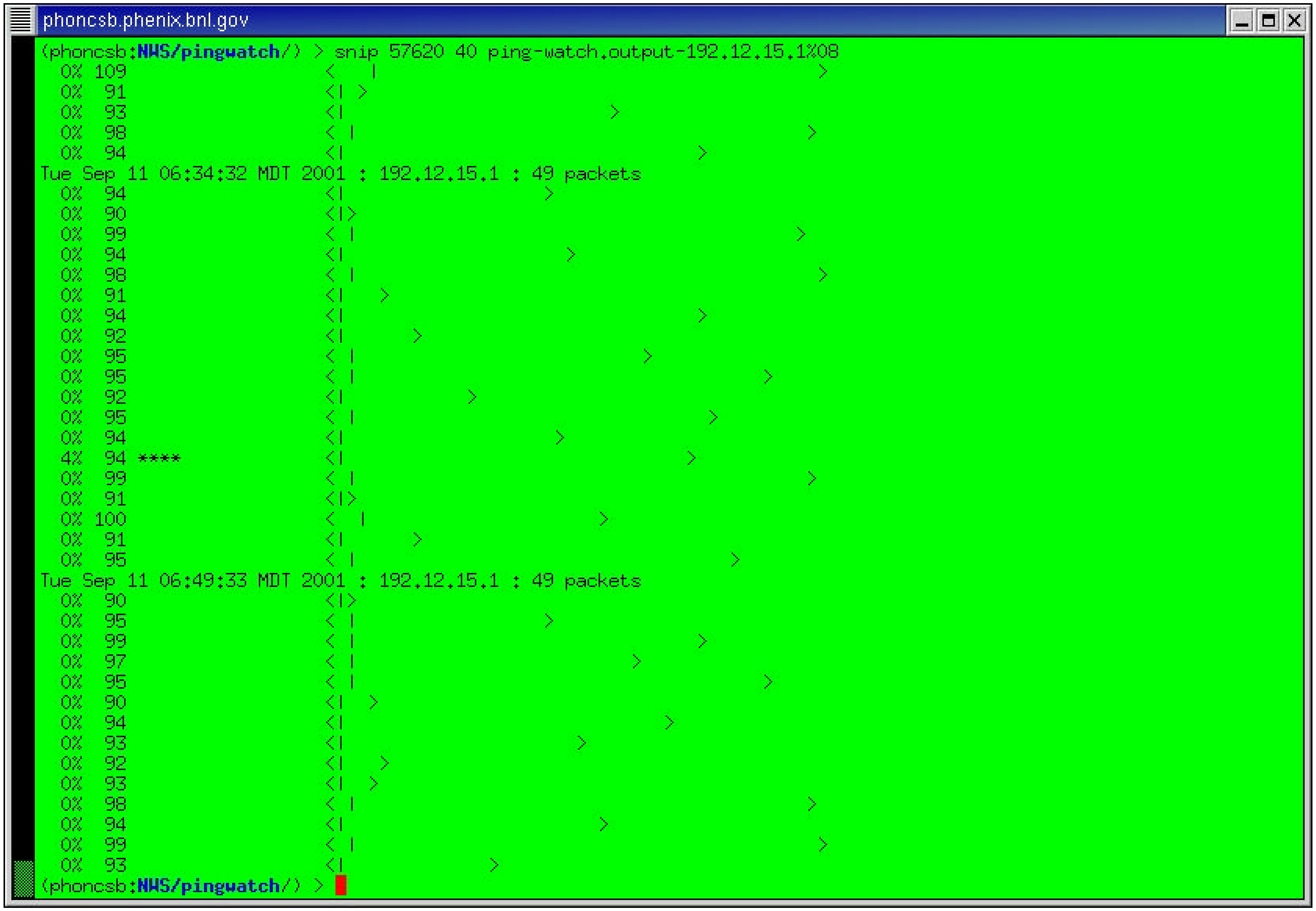
```
             Packet loss rate (%)
             vvv     vvvvvvvvvvvvvvv

[A LINE:]    15% 111 ***************   <   |        >

                 ^^^                   ^   ^        ^
           Avg latency (ms)           min avg      max latency
```

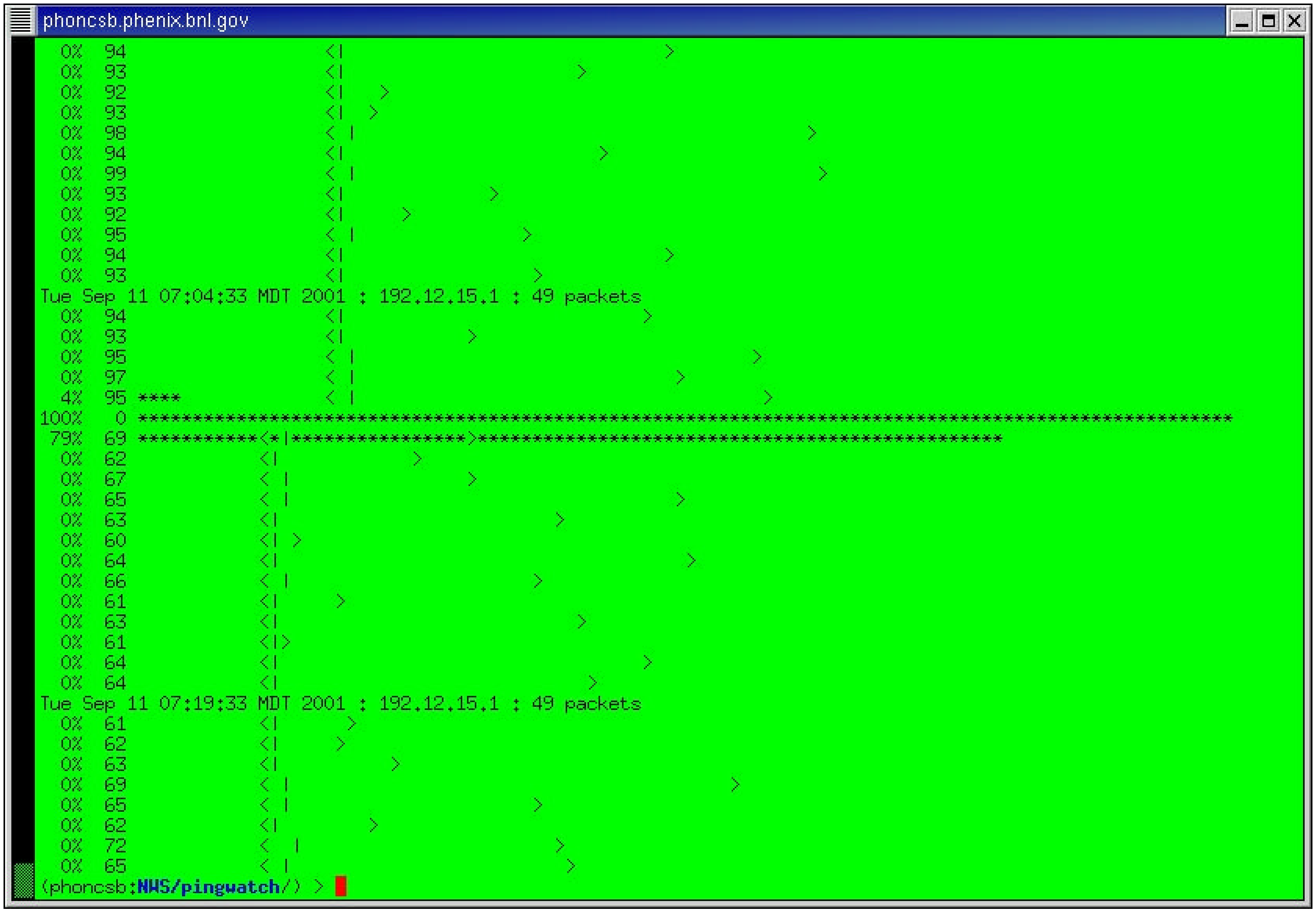



hours
BNL+
LANL+
AXION+
ASTRO+



phoncsb.phenix.bnl.gov

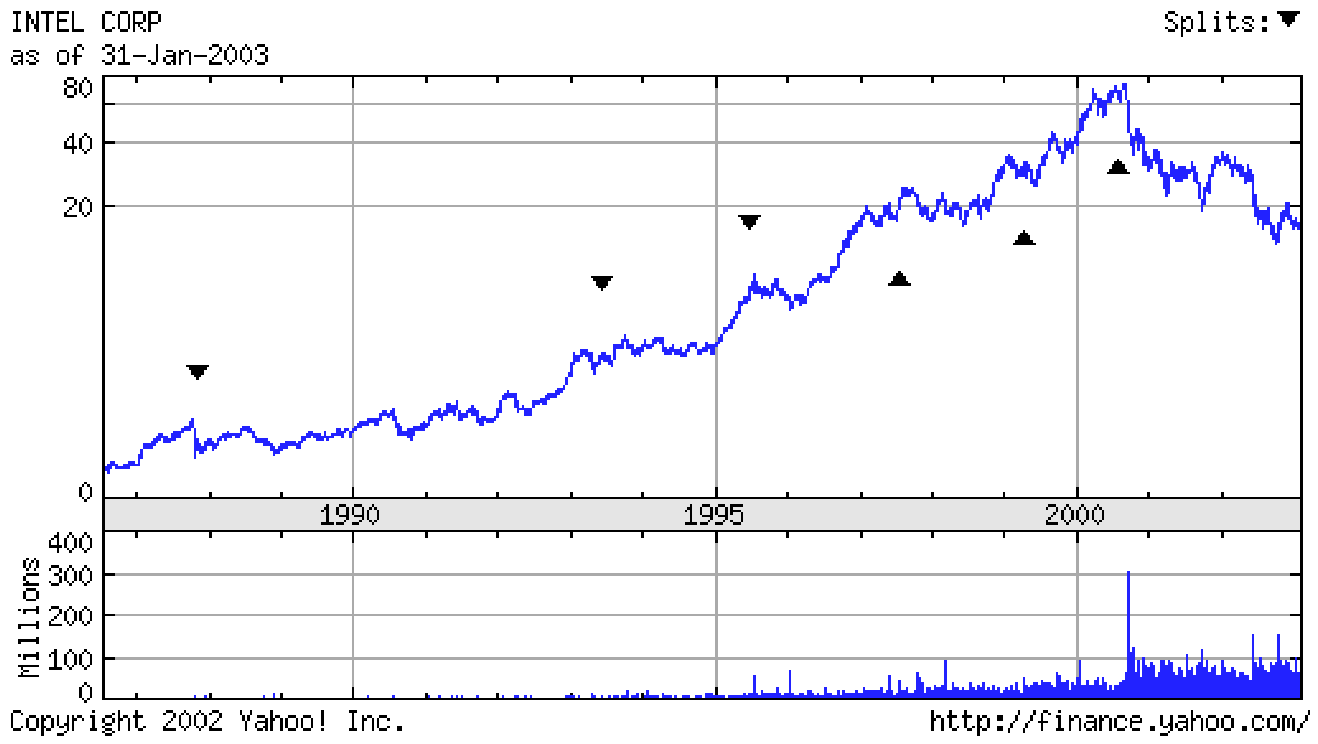
```
(phoncsb:NWS/pingwatch/) > snip 57620 40 ping-watch.output-192.12.15.1%08
  0% 109                  <   |                                        >
  0%  91                  <| >
  0%  93                  <|                        >
  0%  98                  < |                                         >
  0%  94                  <|                                >
Tue Sep 11 06:34:32 MDT 2001 : 192.12.15.1 : 49 packets
  0%  94                  <|                  >
  0%  90                  <|>
  0%  99                  < |                                        >
  0%  94                  <|                    >
  0%  98                  < |                                          >
  0%  91                  <|   >
  0%  94                  <|                                >
  0%  92                  <|      >
  0%  95                  < |                          >
  0%  95                  < |                                     >
  0%  92                  <|           >
  0%  95                  < |                                >
  0%  94                  <|                   >
  4%  94 ****             <|                               >
  0%  99                  < |                                         >
  0%  91                  <|>
  0% 100                  <  |                     >
  0%  91                  <|      >
  0%  95                  < |                                  >
Tue Sep 11 06:49:33 MDT 2001 : 192.12.15.1 : 49 packets
  0%  90                  <|>
  0%  95                  < |                 >
  0%  99                  < |                               >
  0%  97                  < |                         >
  0%  95                  < |                                     >
  0%  90                  <|  >
  0%  94                  <|                             >
  0%  93                  <|                     >
  0%  92                  <|   >
  0%  93                  <|  >
  0%  98                  < |                                         >
  0%  94                  <|                       >
  0%  99                  < |                                          >
  0%  93                  <|             >
(phoncsb:NWS/pingwatch/) >
```



```
  0%  94                  <|                             >
  0%  93                  <|                     >
  0%  92                  <|   >
  0%  93                  <|  >
  0%  98                  < |                                         >
  0%  94                  <|                       >
  0%  99                  < |                                          >
  0%  93                  <|             >
  0%  92                  <|     >
  0%  95                  < |               >
  0%  94                  <|                             >
  0%  93                  <|                 >
Tue Sep 11 07:04:33 MDT 2001 : 192.12.15.1 : 49 packets
  0%  94                  <|                           >
  0%  93                  <|           >
  0%  95                  < |                                    >
  0%  97                  < |                             >
  4%  95 ****             < |                                     >
100%   0 ****************************************************************************************************
 79%  69 ***********<*|****************>************************************************
  0%  62            <|            >
  0%  67            < |                >
  0%  65            < |                                   >
  0%  63            <|                         >
  0%  60            <| >
  0%  64            <|                                     >
  0%  66            < |                      >
  0%  61            <|     >
  0%  63            <|                           >
  0%  61            <|>
  0%  64            <|                                 >
  0%  64            <|                            >
Tue Sep 11 07:19:33 MDT 2001 : 192.12.15.1 : 49 packets
  0%  61            <|      >
  0%  62            <|     >
  0%  63            <|          >
  0%  69            < |                                        >
  0%  65            < |                      >
  0%  62            <|        >
  0%  72            <  |                       >
  0%  65            < |                         >
(phoncsb:NWS/pingwatch/) > 
```

5. Forecasting

To generate a forecast, a Forecaster process requests the relevant measurement history from a Persistent State process. Recall that persistent state is stored as a circular queue by Persistent State processes. If the state is being continually updated by a Sensor, the most recent data will be present when a Forecaster makes its request. Ordered by time stamp, the measurements may then be treated as a time series for the purposes of forecasting.



An NWS Forecaster works only with time stamp-measurement pairs, and does not currently incorporate any modeling information that is specific to a particular series. Instead, it applies a set of forecasting models to the entire series and dynamically chooses the forecasting technique that has been most accurate over the recent set of measurements.

The advantage of this adaptive approach is that it is ultimately non-parametric and, as such, can be applied to any time series presented to the Forecaster. While the individual forecasting methods themselves may require specific parameters, we can include different fixed parameterizations of a particular method with the assurance that the most accurate parameterization will be chosen. ✍ **Promotes extensibility**

When a forecast of a future value is required, the Forecaster makes predictions for each of the existing measurements in the series. Every forecasting model generates a prediction for each measurement, and a cumulative error measure is tabulated for each model. The model generating the lowest prediction error for the known measurements is then used to make a forecast of future measurement values.

5.1 Example Forecasting Results

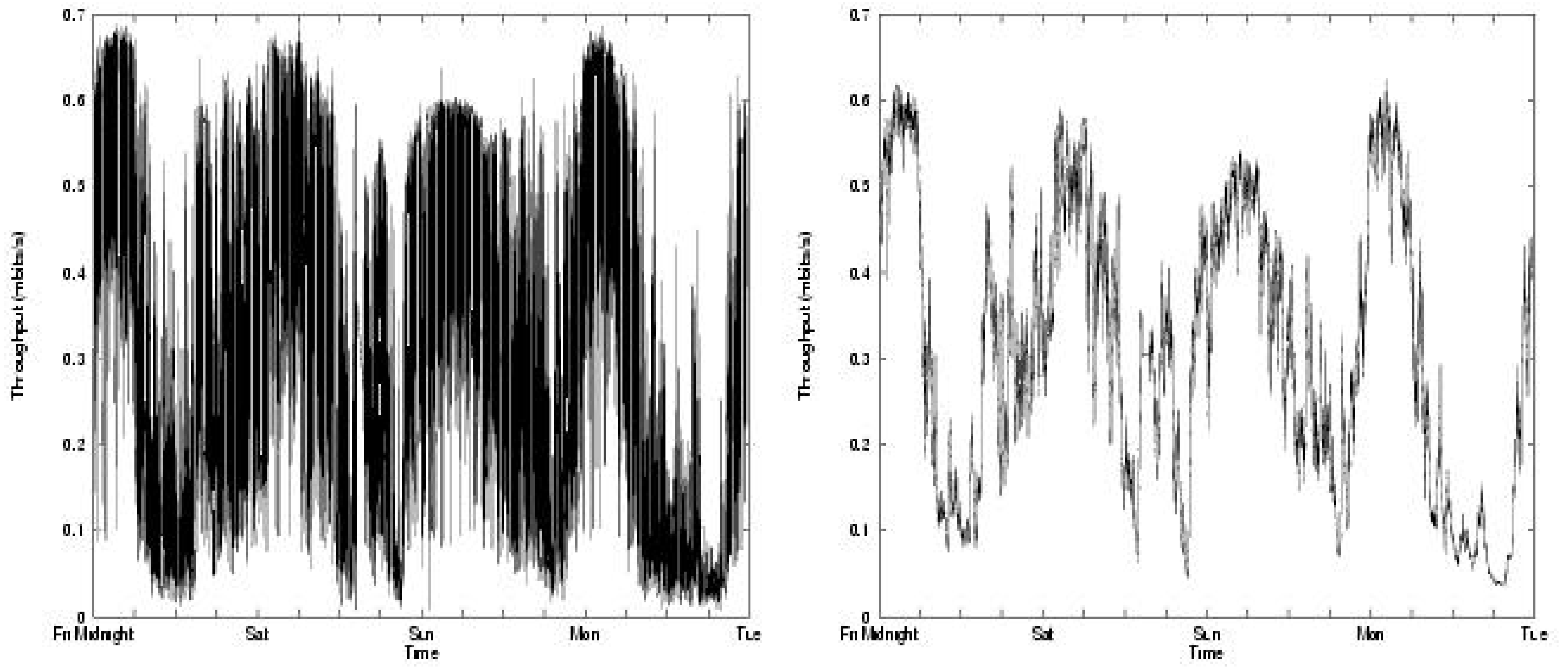
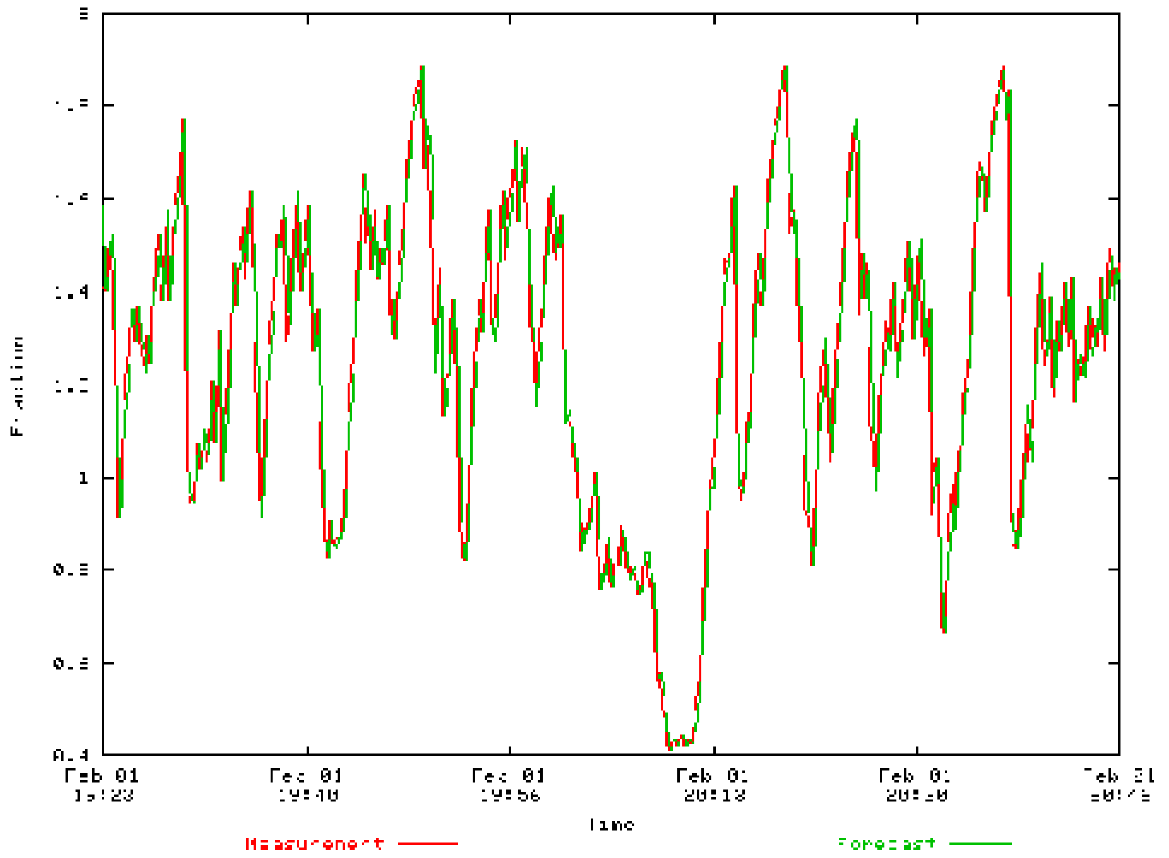


Fig. 3. The left graph shows four days of bandwidth measurements between UC Santa Barbara and Kansas State University. The right graph shows the corresponding NWS forecast values.

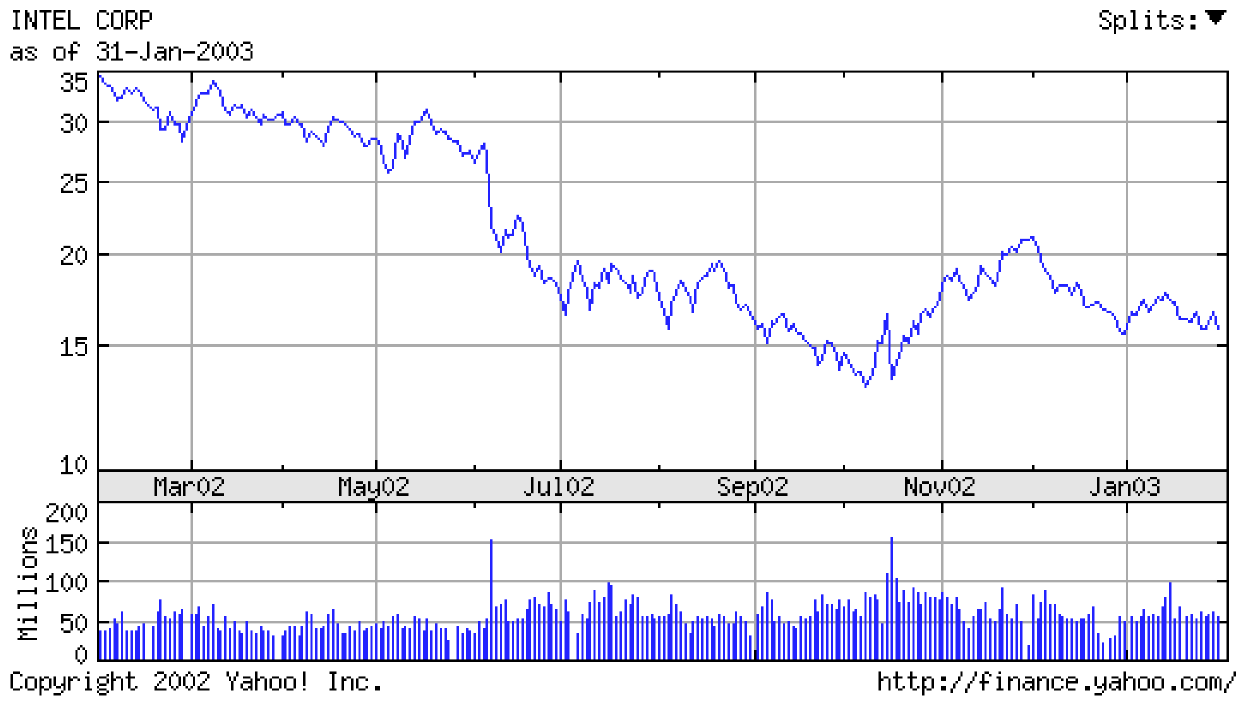
NWS Time Series Query

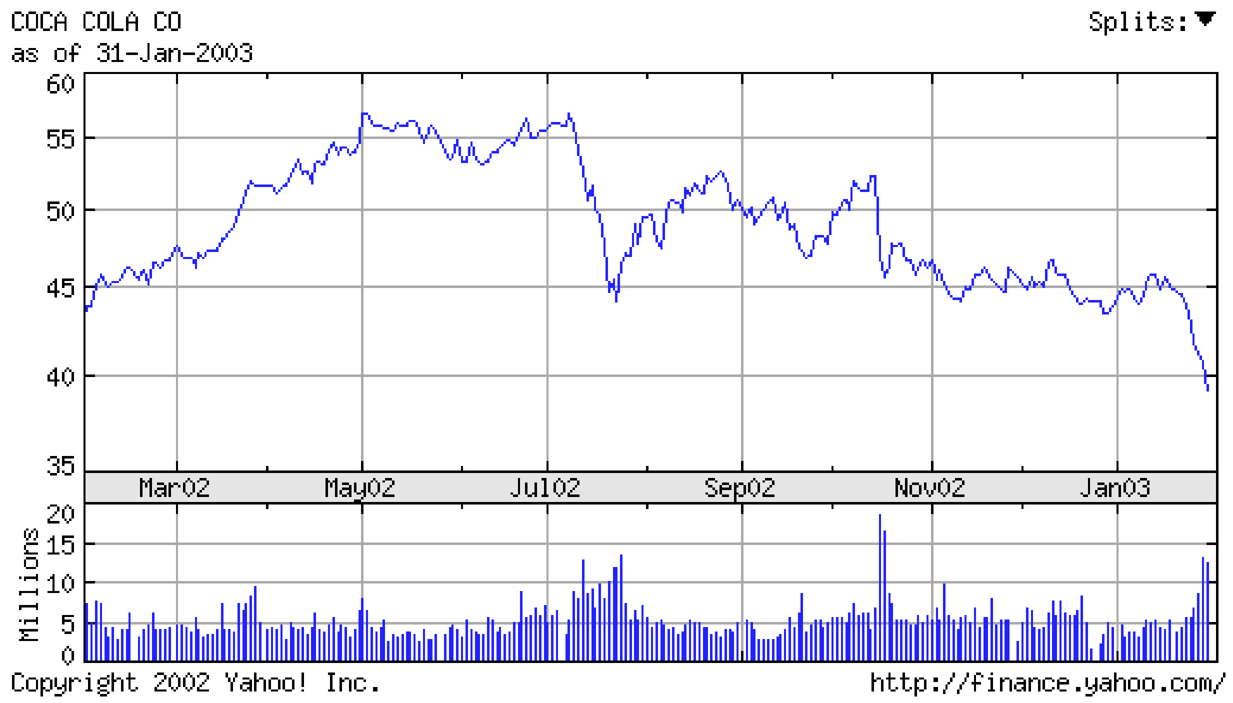
Resource: **currentCpu**
Source: **pompone.cs.ucsb.edu:8060**



Get data | Get postscript

NETWORK WEATHER SERVICE

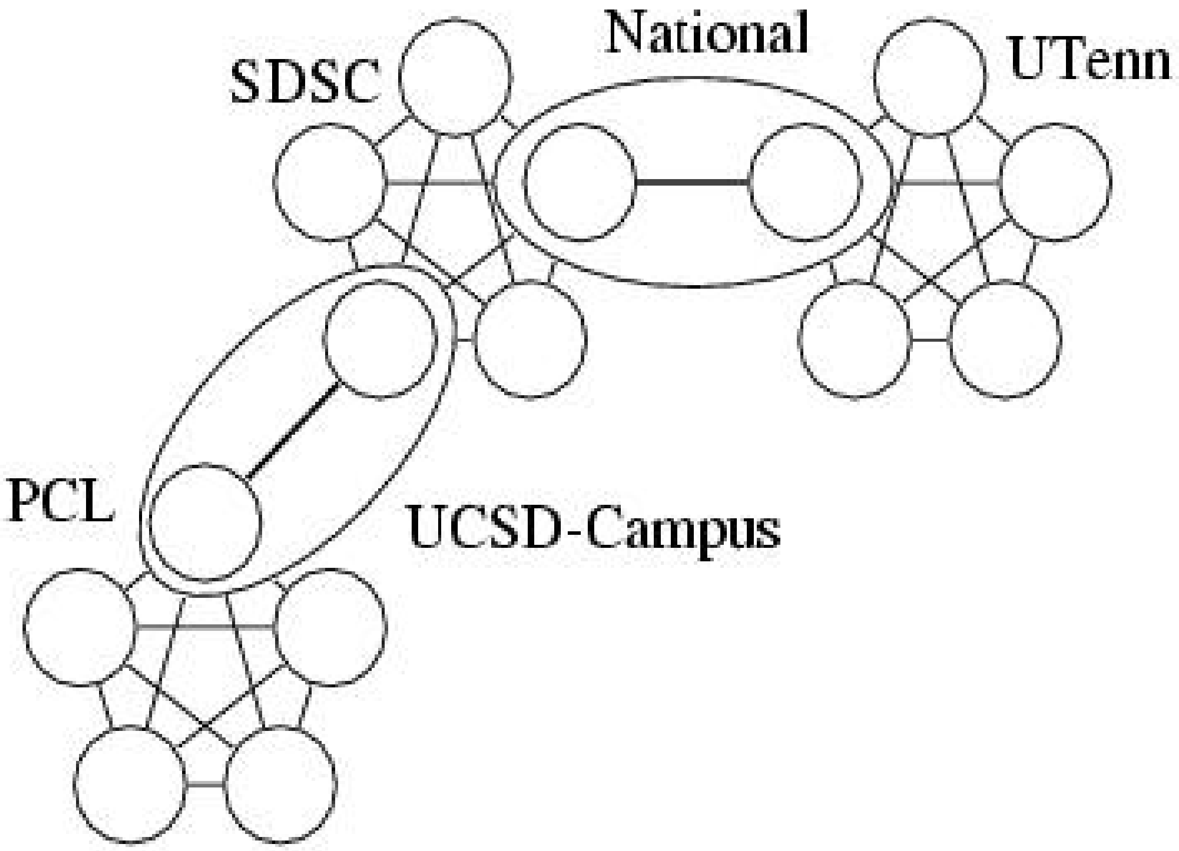

INTEL CORP
as of 31-Jan-2003
Splits:▼
35
30
25
20
15
10
Mar02
May02
Jul02
Sep02
Nov02
Jan03
Millions
200
150
100
50
0
Copyright 2002 Yahoo! Inc.
http://finance.yahoo.com/


COCA COLA CO
as of 31-Jan-2003
Splits:▼
60
55
50
45
40
35
Mar02
May02
Jul02
Sep02
Nov02
Jan03
Millions
20
15
10
5
0
Copyright 2002 Yahoo! Inc.
http://finance.yahoo.com/

7 Sensor Control

To make the system long-lived despite the lossy network connections and intermittent machine failures that occur in any large distributed setting, the NWS relies on adaptive and replicated control strategies. In particular, the Sensors use adaptive time-out discovery and a distributed leader election protocol [15] to remain stable while, at the same time, limiting the load they introduce.

all-to-all network Sensor communication would consume a considerable amount of resources (both on the individual host machines and on the interconnection network) if it were run asynchronously using the entire network Sensor population. The possibility that Sensor probes would collide and thereby measure the effect of Sensor traffic increases quadratically with the number of Sensors. To avoid Sensor contention and to provide a scalable way to generate all-to-all network performance measurements, the network Sensors are organized as a hierarchy of Sensor sets called **cliques**.



We may choose (and we can reconfigure dynamically) to have sub-cliques . if a new site wishes to join the National clique, its representative Sensor can be added dynamically. Further, since the cliques are independent, it is possible to impose different "virtual" hierarchies over the same Sensor population.

To reduce contention within a clique, only a single clique member conducts experiments at any given time. This policy is implemented by passing a clique token among member Sensors.

Once the token has visited all Sensors in the clique, it is returned to the initiating Sensor (the **leader**) which is then responsible for re-initiating it. The periodicity with which the clique leader re-initiates the token controls the periodicity with which each Sensor conducts its probes.

Each Sensor then calculates a local time-out based on the last time it held the token and the time-out that the leader has determined. If the local time-out expires before a Sensor receives the token again it assumes that either the token has been lost or the network has partitioned. It then generates a new token, marks itself as its leader, and initiates it into the system.

In this way, if the token has been lost due a Sensor failure, a new one will be initiated.

if the network partitions into disjoint sets, at least one token will be started in each set when the time-out occurs.

To prevent multiple tokens from consuming network resources indefinitely, tokens are sequenced, and any Sensor encountering an old token discards it rather than propogating it to the next Sensor.

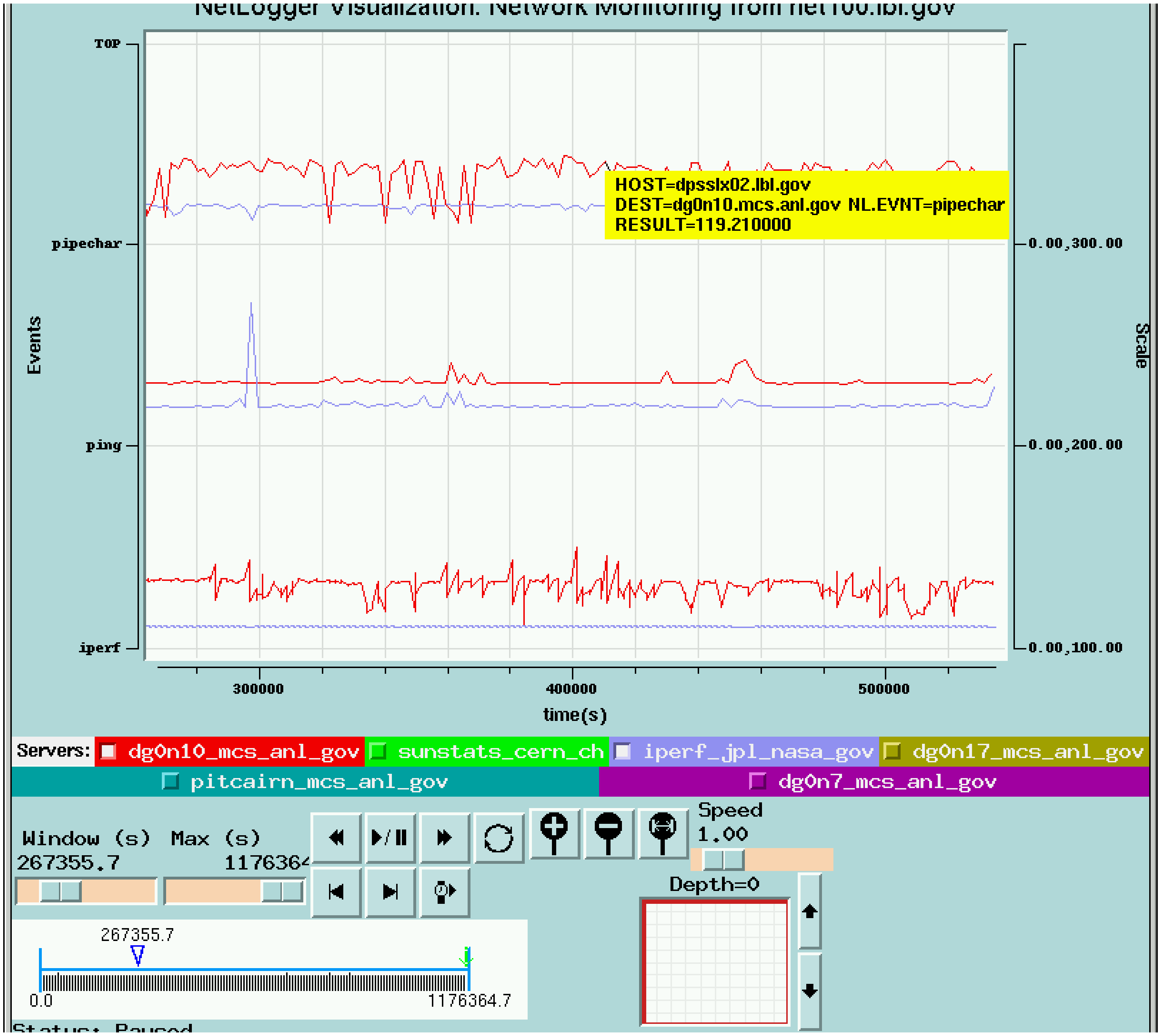
7.1 Adaptive Time-out Discovery

The stability of the token protocol depends on the clique leader's ability to determine when the token should be timed out.

The Sensors, therefore, require a prediction of what the time-out value should be, given the performance variations of the network. To make this prediction, the Sensors use the prediction techniques that are integrated with the Forecasters. The clique leader passes a time series of circuit times to a local Forecaster interface and receives back a predicted circuit time and an estimate of the variance

NASA IPG and the NWS

- NWS Interface to Globus
- NWS application forecasting library – *dynamic benchmarking*
- Archival storage facility based on Netlogger
- AppLeS enhancements, support, and good will



HOST=dpsslx02.lbl.gov
DEST=dg0n10.mcs.anl.gov NL.EVNT=pipechar
RESULT=119.210000

Outline:

\ My history; my monitoring stuff; cf. previous / continuing HEP monitoring infrastructure.

\ My old Coke / Intel comparison forecast

o History / people of the NWS project

o NWS is distributed as part of…

o

o

\ Take a look at including a bit of Jenny Schopf’s stuff on prognostication.

o

o